IUT de Bayonne

Table des matières

Philippe.Roose@iutbayonne.univ-pau.fr

1 Introduction

1.1 Origine de ce langage

PHP, écrit en C, est né avec le site de Rasmus Lerdof. Il désirait mettre son CV en ligne et garder une trace des utilisateurs. Dès le départ, il supportait les requêtes SQL, et devant le nombre de personnes intéressées, il a décidé de mettre en ligne la version 1.0 de PHP (Personal Home Page). Cette version permettait de plus de contrôler les saisies des utilisateurs et de remplacer certaines commandes.
Devant le succès et le nombre de requêtes des programmeurs, en utilisant Bison/Yacc et mis en œuvre PHP 2.0. Cette version ajoutera des structures conditionnelles, des boucles et encore bien d'autres améliorations. Elle permettait également au programmeur d'intégrer des instructions de programmation puissantes à l'intérieur de leur code HTML.

Le script PHP est maintenant directement compilé dans le serveur Web (Apache par exemple), les instructions sont donc exécutées sur le serveur lui même.

Ce pauvre Rasmus devenant un peu débordé, il a fait appel à d'autres programmeurs de renom et le 6 juin 1998 est né PHP 3.0, ce dont nous allons parler maintenant.

1.2 Atouts

On peut se demander pourquoi utiliser du PHP alors qu'il existe bien d'autres technologies et en particulier le Perl (le plus populaire) pour les solutions CGI. Microsoft propose les ASP (Active Server Pages) avec son serveur Web IIS (Internet Information Server). Il existe d'autres solutions plus ou moins propriétaires, à la fois gratuites et payantes.
Le choix de PHP se réalise vite : c'est le meilleur ! Il intègre tous les avantages des autres solutions. Son code est rapide à produire et à exécuter. Il tourne sur différents serveurs (IIS, WebTen, …) et différents systèmes d'exploitation (UNIX, Windows et Macintosh). Et puis, il possède un autre avantage, qui, s'il n'est pas son principal, en est un de plus : il est gratuit.

1.3 PHP dans une page Web

De manière classique, voici l'éternel « Hello World » dans sa version PHP3. La commande *print* (ou *echo*) permet d'afficher le texte entre parenthèses.

```
<HTML><BODY>
<?php
print "Hello World<P>";
?>
</BODY></HTML>

affichera :
1.3.1.1.1.1    Hello World
```

Le code PHP3 est placé directement dans le code source HTML, il est encadré par deux balises spéciales qui sont destinées à être reconnues par l'interpréteur PHP3. On aurait pu tout aussi bien pu utiliser les balises <SCRIPT LANGUAGE="php"> et </SCRIPT> comme on le fait pour du JavaScript.

Attention,
Il ne faut pas confondre PHP3 et JavaScript. Il s'agit de deux choses différentes. PHP3 est un langage de script destiné à être exécuté par le serveur, alors que le JavaScript est chargé et exécuté dans le navigateur, donc « coté client ». PHP3 est donc comparable aux ASP de Microsoft ou aux CGI « standards » dans le sens où il s'exécutent tous les deux « côté serveur ».

Enfin, il est également possible d'utiliser les tags sous leur forme courte (<? et ?>), mais cette possibilité n'est pas activée par défaut. Il faudra donc modifier le paramétrage de PHP3 sur la machine serveur.

Quand on affiche dans le navigateur le source HTML d'une page générée par PHP3, on ne voit que le code HTML résultant. Si vous regardez le source de la page précédente dans votre navigateur, vous obtiendrez ça :

```
<HTML><BODY>
Hello World<P>
</BODY></HTML>
```

2 Variables, opérateurs et expressions

2.1 Identifiants

PHP propose trois types de données : les entiers, les décimaux, et les chaînes de caractères. Lors de l'identification des variables PHP respecte la casse. Ainsi, *maVariable* est différents que *MaVariable*. Une variable est toujours précédée du signe $ que les amateurs de Shell connaissent bien.

2.2 Types de données

En fait, quand on utilise du PHP, on ne s'occupe pas de distinguer le types des variables. Elles prennent le type des données qui leur sont affectées. Néanmoins, il est possible de contourner ceci si nécessaire en utilisant la fonction *settype* qui attribue un type spécifique à une variable. Il existe une autre solution par l'utilisation des fonctions *intval, doubleval* et *strval* permettant de réaliser des conversions.

Exemple

```
< ?
$temperature = « 30° » ;
settype($temperature, « double ») ;
print « Double : $temperature <BR>\n»
settype("$temperature", "integer") ;
print « Integer : $temperature <BR>\n»
settype("$temperature", "string") ;
print « String : $temperature <BR>\n»

$temperature = « 30° » ;
print " <BR>String <BR>";
print "strval($temperature)";
print "<BR>Double <BR>";
print "doubleval($temperature)";
print "<BR>Integer <BR>";
print "intval($temperature)";

$foo = "0";  // $foo est une chaine
$foo++;      // $foo est une chaine
$foo += 1;   // $foo est maintenant un entier
$foo = $foo + 1.3;  // foo est maintenant un double
$foo = 5 + "10 Little Piggies"; // $foo is integer (15)
$foo = 5 + "10 Small Pigs";     // $foo is integer (15)
?>
```

Il est également possible de réaliser ceci de manière implicite en utilisant le CAST vu en C. Ainsi, *intval($temperature)* et *(integer) $temperature* sont identiques.

2.3 Portée des variables

A l'inverse d'autres langages comme le C, il est inutile en PHP de déclarer les variables avant de les utiliser. C'est l'une des caractéristiques des langages interprétés.
Il est à noter que comme dans la plupart des langages, la portée des variable égale celle du bloc où elle a été utilisée pour la première fois (ce qui équivaut à une déclaration). Dans les cas où une variable déclarée dans une

fonction est nécessaire au niveau global, il est possible de lui étendre sa portée de manière à ce qu'elle continue d'exister une fois la fonction terminée. On pourra pour cela la déclarer comme globale :

Exemple
```
function maFonction( ) {
  global maVar ;
   …
}
```

Une autre manière plus élégante est d'utiliser le mot clé *static* qui dans la plupart des cas permet de s'affranchir de l'utilisation peu élégante de *global*.

2.3.1.1.1.1 Exemple
```
static mavar ; // static à le même comportement qu'en C
```

2.4 Assignation de valeurs aux variables

Lorsque l'on assigne une valeur à une variable (=), son type change pour correspondre au type de données qui lui est affecté. Ce fonctionnement est l'inverse du C qui lui essaie de convertir la donnée au type de la variable.

Exemple
```
$machaine1 = « Gérard » ;
$machaine2 = « Dupont \n» ;
integer var1 ;
$var1 = 12 ;
$var1 = -45 ;
double var1 ;
$var1 = -3.14
```

le signe \ permet d'éviter l'affichage du caractère suivant :

\ ''	Guillemets doubles
\\	Caractère antislash
\n	Nouvelle ligne
\r	Retour chariot
\t	Tabulation
\x00 - \xFF	Caractères hexadécimaux

Il est à noter qu'il est tout à fait possible d'inclure une variable dans une chaîne de caractères :

Exemple :
```
print « Je suis : $machaine1 <BR> » ; // Affichera « Je suis Gérard »
```

2.5 Constantes

Il existe un certain nombre de constantes crées automatiquement par PHP (nous en verrons quelques unes plus loin). Il est également possible de définir ses propres constantes avec *define(nomconstante, valeur constante)*. En réalité, une constante est une variable dont le contenu ne peut être paramètré qu'une seule fois. Contrairement aux variables, les constantes sont par défaut sensibles à la casse. Associé à la fonction *define*, une autre fonction *defined(''nom constante'')* peut être utilisée afin de tester l'existence d'une constante (elle retournera *true* ou *false* selon le cas).

Exemple
```
define("MACONSTANTE", "Hello World") ;
if (defined(''MACONSTANTE'')) {
  print ''La valeur de ma constante est : ''.MACONSTANTE //Attention, on accède au contenu sans le $.
}
```

2.6 Opérateurs logiques, binaires, autres

Nous n'allons pas décrire l'utilisation de chacun des opérateurs, ils sont quasi-identiques au C.

Opérateurs arithmétiques :

+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Module
++	Incrémentation
--	Décrémentation

Opérateurs logiques

<	Inférieur à
>	Supérieur à
<=	Inférieur ou égal à
>=	Supérieur ou égal à
==	Égal à
!=	Différent
AND &&	Et
OR ||	Ou
XOR	Ou exclusif
!	Sauf

Autres opérateurs
Le . permet de faire une concaténation (*print $var1.$var2* affichera le contenu de var1 concaténé à var2).

3 Instructions de contrôle

Pas de surprises en PHP pour l'utilisation des instructions de contrôle. Elles sont identiques à celles déjà vues en C.

3.1 If, then, else

```
if (expression) {
   Bloc si expression vraie
}
else {
        Bloc si expression fausse
}
```

ou bien

```
if (expression 1) [
  Bloc si expression vraie
}
else if (expressio2) {
  Bloc si expression2 vraie
}
else {
  Bloc si expression2 est fausse
}
```

3.2 Switch

```
switch (expression) {
  case exprt1: …
  default : …
}
```

Exemple d'utilisation de *switch*

```
<?php

$note=11;
$extension="fr";

switch($note) {
  case $note>10:
     echo ("Bac obtenu");
     break;

  case $note>=12:
     echo ("Bac obtenu avec mention AB");
     break;

  case $note>=14:
     echo ("Bac obtenu avec mention B");
     break;

  case $note>=16:
     echo ("Bac obtenu avec mention B");
     break;

default :
     echo ("Bac non obtenu");
     break;
}

print "<br><br>";

switch($extension) {
  case "fr" : $pays="France";
           break;
  case "es" : $pays="Espagne";
           break;
  case "Be" : $pays="Belgique";
           break;
  default   : $pays="Inconnu";
           break;
}
print "Pays visité : $pays";


php?>
```

Il n'est pas nécessaire de mettre des accolades pour chacun des cas.

3.3 Boucles for, do…while, for

```
for (initialization; expression; incrémentation) {
…
```

```
}
```

```
while (expression) {
…
}
```

```
do {
…
} while (expression);
```

3.4 Instructions break et exit

Les instructions *break* et *exit* ont le même comportement qu'en C.

4 Fonctions

Tout comme dans la plupart des langages de programmation, il est également possible d'écrire ses propres fonctions en PHP.

4.1 Déclarations

La syntaxe est proche de ce qui est réalisé par exemple en C :

```
function nomFonction (paramètres) {
  Corps de la fonction
}
```

4.2 Arguments

Voici par exemple une fonction qui permet d'afficher en gras une chaîne de caractères :

```
function bold ( $machaine ) {
  print ''<B>'' . $machaine . '' </B> '' ;
}
```

Comme cela est possible en C++, nous pouvons définir des paramètres par défaut :

```
function afficher($maChaine, $couleur = ''black'') {
  print ''<FONT COLOR=\ ''$Color\ ''> $maChaine </FONT>'' ;
}
```

Si le deuxième paramètre n'est pas précisé lors de l'appel, la fonction choisira le noir.
Par défaut, les passage de paramètre sont réalisés par recopie. Il est possible de réaliser un passage de paramètres par référence (l'original donc !) en précédent le nom du paramètre pas & dans la déclaration de la fonction, à l'instar de ce qui se fait en C.

4.3 Retour de paramètres

Il est également possible de faire retourner une valeur à la fonction :

```
function retourneBold ( $machaine ) {
  $chaineTmp = ''<B>'' ;
  $chaineTmp .= $machaine ; // Le point permet de concaténer
  $chaineTmp .= ''</B>'' ;

return ($chaineTmp) ;
}
```

5 Tableaux

Pas de surprises avec l'utilisation des travaux, l'utilisation est tout à fait classique.

5.1 Tableaux à une dimension

```
< ?
  $bab[0] = “Biarritz”;
  $bab[1] = ”Anglet”;
  $bab[2] = ”Bayonne”;

print “Je vis à $bab[1]. <BR>’’ ;
?>
```

Dans les cas où l’on ne désire pas de préoccuper de mentionner les index, PHP le fera à votre place :

```
  $bab [ ] = ‘’Biarritz”;
  $bab [ ] = ”Anglet”;
  $bab [ ] = ”Bayonne”;
```

Si l’on désire savoir combien de valeurs ont été entrées dans un tableau, on utilisera la fonction *count(tableau)* :

```
$limiteIndex = count ($bab) ;
```

et pour les afficher :

```
for ($index = 0 ; $index < $limiteIndex ; $index++) {
   print (“$bab[$index] <BR>”);
}
```

Il est également possible d'indexer les tableaux autrement qu'avec des entiers :

```
print "Exemples de tableaux : <br>";
$capitale[fr]="Paris";
$capitale["es"]="Madrid";

print "La capitale de l'Espagne est : $capitale[es]";
```

A noter que les guillemets ne sont pas nécessaires à l'initialisation, mais ils ne doivent pas être mis à l'utilisation…autant dire qu'il est inutile de les mettre !

5.2 Initialisation

Il est également possible de déclarer un tableau avec ses initialisations. Ce type de tableau est appelé associatif :

```
< ?
$mois=array(2=> ‘’Janvier’’, ‘’Février’’, ‘’Mars’’,
‘’Avril’’, ‘’Mai’’, ‘’Juin’’, ‘’Juillet’’, ‘’Août’’, ‘’Septembre’’, ‘’Octobre‘’, ‘’Novembre’’, ‘’Décembre’’) ;
?>

print ‘’Mois de $mois[5]’’ ; affichera « Mois de Avril ».
```

L’opérateur **=>** permet ici de faire démarrer le tableau à l’index 1 et non pas 0 comme c’est le cas par défaut.

Enfin , il n'est pas nécessaire d'avoir des tableaux avec des indices numériques successifs.

```
$tabbizarre[12] = "pas bien grand";
$tabbizarre[1025] = "plutot grand";
$tabbizarre[15215] = "très grand";
```

Afin d'accéder aux éléments, un ensemble de fonctions sont définies :

```
print "Key : ".key($tabbizarre)." <br>";
print "Current : ".current($tabbizarre)." <br>";
print "Next : ".next($tabbizarre)." <br>";
print "Prev : ".prev($tabbizarre)." <br>";

> Key : 12
> Current : pas bien grand
> Next : plutot grand
> Prev : pas bien grand
```

Afin de réaliser un parcours efficace de ce genre de tableaux, deux fonctions sont bien utiles. La fonction *each* permet de renvoyer uniquement les éléments du tableau contenant les données. La fonction *list* ($indice, $valeur) récupère l'indice et la valeur de l'élément.

```
while (list($indice,$contenu) = each ($tabbizarre))
  print "$contenu <br>";
```

5.3 Tri de tableaux

La fonction *sort()* permet de trier dans l'ordre alphabétique le contenu d'un tableau. Sa cousine *ksort()* fait de même mais dans l'ordre inverse. Les deux autres *asort()* et *arsort()*font la même chose mais pour des tableaux avec indice chaînes.

```
$tabatrier[] = "albert";
$tabatrier[] = "george";
$tabatrier[] = "bacus";

sort($tabatrier);
while (list($indice,$contenu) = each ($tabatrier))
  print "$contenu <br>";

print "<br>";

rsort($tabatrier);
while (list($indice,$contenu) = each ($tabatrier))
  print "$contenu <br>";


> albert
> bacus
> george

> george
> bacus
> albert
```

5.4 Tableaux à dimensions multiples

L'utilisation de tableaux à dimensions multiples est là encore semblable aux autres langages. Ainsi si un tableau représente un échiquier, nous accéderons aux valeurs en faisant par exemple :

```
$echiquier[5][4] ;
```

6 Entrées/Sorties

PHP étant un langage de script, il doit pouvoir être capable de communiquer avec son environnement. Les scripts PHP étant exécutée dans le contexte d'un navigateur Web, il est important de se rappeler que toute sortie vers le navigateur sera interprété comme du texte HTML.

6.1 Envoi vers le navigateur

Trois fonctions permettent d'envoyer du texte vers le navigateur : *echo*, *print* et *printf*. L'intérêt principal de cette dernière fonction est, nous le verrons plus tard, la possibilité de spécifier un format de sortie afin d'éviter d'envoyer la valeur telle quelle.

6.2 Récupération des données d'un formulaire

Si l'envoi de données vers le navigateur est simple en utilisant les trois fonctions décrites précédemment, la récupération d'informations en provenance du navigateur l'est un peu moins.
Le langage HTML permet la récupération de données d'un formulaire : champs/zones de texte, listes déroulantes, cases à cocher, …
PHP transforme tous les champs d'un formulaire en variables ce qui permettra de modifier éventuellement leur contenu.
Soit le formulaire suivant :

```
<FORM ACTION="/php/myprogramme.php"   METHOD="POST">
Saisissez votre nom :
<INPUT TYPE="TEXT" NAME="NOM" SIZE="20"> MAXSIZE="50" ><BR>
<INPUT TYPE="SUBMIT" VALUE="Envoyer la requête">
</FORM>
```

Dans le fichier *myprogramme.php*, il sera possible de récupérer la valeur saisie dans le champ NOM de la manière suivante :

```
$nom = $_POST['NOM'] ;
```

Attention, vous trouverez dans de nombreux scripts une récupération de variable de formulaire directement (print ''$NOM '') par exemple. Cette technique employée jusqu'à peu montre ses limites avec les dernières versions de PHP (et quel que soit le mode de passage des paramètres : POST, GET, cookie, session, …). La différence est que depuis la version 4.2.0, l'option register_globals du php.ini qui était positionnée par défaut à ON est à OFF. Donc, autant prendre de bonnes habitudes et utiliser la nouvelle « technique ». Pour vérifier la version de PHP, utilisez la fonction phpinfo() qui retourne une multitude d'informations. Ca vaut le coup d'œil.

Voici un récapitulatif des variables et des changement selon les versions :

Avant la version 4.10	Après la version 4.10
$HTTP_GET_VARS	$_GET
$HTTP_POST_VARS	$_POST
$HTTP_POST_FILES	$_FILES
$HTTP_COOKIE_VARS	$_COOKIE
$HTTP_SESSION_VARS	$_SESSION
$HTTP_ENV_VARS	$_ENV
$HTTP_SERVER_VARS	$_SERVER

Pour récupérer facilement les variables, vous pouvez utiliser la fonction PHP extract. Elle va exporter votre tableau associatif et créer une variable pour chaque clé du tableau.

```
extract($_POST,EXTR_OVERWRITE);
```

Cette fonction va créer une variable pour chaque clé du tableau associatif $_POST.Si on a :

- $_POST['nom']
- $_POST['prenom']
- $_POST['age']

La fonction extract() va créer les variables suivantes :

- $nom
- $prenom
- $age

Type	signification
EXTR_OVERWRITE	Écrase les variables existantes
EXTR_SKIP	N'écrase pas les variables existantes
EXTR_PREFIX_SAME	Si une variable existe déjà, une nouvelle variable est créée avec un préfix donné en 3ème argument à la fonction
EXTR_PREFIX_ALL	Crée de nouvelles variables avec le préfix passé en 3ème argument pour toutes les clés du tableau
EXTR_PREFIX_INVALID	Crée de nouvelles variables avec le préfix passé en 3ème argument pour les noms de variable invalides (par exemple $1)

6.3 Récupération des variables d'environnement

Tableau $_SERVER /HTTP_SERVER_VARS

- **PHP_SELF** : Le nom du fichier du script en cour d'exécution, par rapport au document root.
- **SERVER_NAME** : Le nom du serveur hôte qui exécute le script suivant. Si le script est exécuté sur un hôte virtuel, ce sera la valeur définie pour cet hôte virtuel.
- **DOCUMENT_ROOT** : La racine sous laquelle le script courant est exécuté, comme défini dans la configuration du serveur.
- **REMOTE_ADDR** : L'adresse IP du client qui demande la page courante.
- **REMOTE_PORT** :Le port utilisé par la machine cliente pour communiquer avec le serveur web.
- **SCRIPT_FILENAME** : Le chemin absolu jusqu'au script courant.
- **SERVER_PORT** : Le port de la machine serveur utilisé pour les communications. Par défaut, c'est '80'. En utilisant SSL, par exemple, il sera remplacé par le numéro de port HTTP sécurisé.
- **REQUEST_URI** : L'URI qui a été fourni pour accéder à cette page. Par exemple : '/index.html'.

Tableau $_FILE/$HTTP_POST_FILES

- **['NOM OBJET FILE']['name'] :** permet de récupérer le nom du fichier sélectionné dans un objet ***file*** du navigateur (exemple, fichier à télécharger).
- **['NOM OBJET FILE']['tmp_name'] :** retourne le nom du fichier temporaire stocké sur le serveur, en attente de transfert à la destination désirée.
- **['NOM OBJET FILE']['size'] :** retourne la taille en octet du fichier.
- **['NOM OBJET FILE']['type'] :** donne le type MIME du fichier.

6.4 Fonction include

Tout comme il est possible de réaliser des inclusion de fichiers en C avec la directive de compilation *#include* il est également possible de le faire en PHP avec la fonction *include(nom de fichier)*. Cette fonction très utile permet une réutilisation maximale de fonctions déjà écrites.

6.5 Lecture/écriture de fichiers

Comme dans la plupart des langages, afin de travailler avec un fichier, il est nécessaire d'ouvrir l'accès au fichier, d'y accéder, puis de le fermer.
Il est possible en PHP d'ouvrir un fichier local (fonction *fopen*) ou distant et le désignant par une URL et un protocole (*http* ou *ftp*).
Les différents modes d'ouvertures sont les suivants :

- *r* : ouverture en lecture seule,
- *r+* : ouverture en lecture/écriture,
- *w* : ouverture en écriture seule. Si le fichier existe déjà, il est effacé, sinon, il est crée (si possible),
- *w+* : ouverture en lecture/écriture. Si le fichier existe déjà, il est effacé, sinon, il est crée (si possible),
- *a* : ouverture en ajout uniquement. Les données sont écrites à la fin du fichier, s'il n'existe pas, il est crée (si possible),
- *a+* : ouverture en lecture et ajout. Les données sont écrites à la fin du fichier, s'il n'existe pas, il est crée (si possible).

Afin de revenir au début du fichier, il est nécessaire d'utiliser la fonction *fseek*. Le premier paramètre est l'identifiant du fichier retourné par *fopen*, le second paramètre est l'offset (décalage en nombre d'octets, à partir du début du fichier - exprimé par un entier).

Nous n'allons pas le décrire ici, mais il est également possible d'accéder à des fichiers en ouvrant un canal ou *pipe* avec la fonction *popen* ou en ouvrant une connexion par *socket* avec *fsockopen*.

Voici un exemple d'ouverture fichier :

```
< ?
$monFichier = fopen(''monfichier.txt'','' w'') ; // ouverture en écriture
if ( !($monfichier)) {
  print('' Impossible de créer le fichier \n'') ;
  exit ;
}

fputs($monfichier, ''ligne 1'') ; // on écrit deux lignes
fputs($monfichier, ''ligne 2'') ;

fclose($monfichier) ; // on ferme le fichier, on libère les ressources

$monfichier = fopen(''monfichier.txt'', ''r'') ; // ouverture en lecture
if ( !($monfichier)) {
  print(« Impossible d'ouvrir le fichier ») ;
  exit ;
}
while ( !feof($monfichier) ) {
  $ligne = fgets($monfichier,255); // 255 caractères max. ou bien fin de ligne.
  print '' $ligne <BR> '' ;
}

fclose ($monfichier) ;

?>
```

On peut également accéder à des fichiers distants en passant par une URL :

```
$url = ''http://wwwbay.univ-pau.fr/~roose/index.shtml'' ;
fopen($url) ;
```

Il est parfois nécessaire de lire un fichier pour l'afficher par la suite dans le navigateur. La fonction *readfile* permet ceci en renvoyant le contenu du fichier vers le navigateur. La valeur retournée est le nombre d'octets lus. Si le nom du fichier commence par **http://** ou **ftp://**, l'accès à ce dernier sera fait via l'un de ces protocoles. Dans le cas où aucun protocole n'est spécifié, l'accès se fait sur le système local (celui-ci sur lequel est hébergé la page PHP correspondante.

En utilisant la fonction *copy*, il est possible de copier un fichier donné en argument vers la destination spécifiée. La fonction retourne *true* si l'opération s'est bien déroulée, *false* sinon. A l'aide de cette fonction, il est par exemple possible de transférer un fichier local (à celui qui accède à la page) vers un serveur.

Il existe toute une série de fonctions associées aux fichiers, comme :

- *file_exists(descripteur de fichier)* : teste l'existence d'un fichier (ne fonctionne pas avec des accès HTTP ou FTP). Attention,
- *filesize('' nom fichier'')* qui retourne la taille en octets du fichier.
- *fseek (descripteur fichier, offset)* : permet de se positionner à l'octet « offset » dans le fichier. Cette fonction retourne 0 si tout c'est bien passé, -1 sinon.

- *chmod(fichier, droits unix)* : change les droits du fichier (à condition que le serveur PHP ai également les droits dessus).
- *is_dir(''nom fichier'')* ; *is_executable(''nom fichier''), is_link(''nom fichier''), is_readable(''nom fichier''), is_writeable(''nom fichier'')* permettant de savoir si un fichier est un répertoire, exécutable, un lien, si les droits en lecture ou écriture sont valides. Lorsque les évaluations sont vérifiées, 1 est retourné, 0 sinon.
- *opendir(''nom répertoire'')* retourne un état de répertoire exploitable par la suite avec *readdir* et *closedir*.
- *readdir(descripteur de répertoire)*, retourne le nom du fichier suivant à partir du descripteur de répertoire obtenu avec *opendir*. On fermera le répertoire avec *closedir* comme on le fait lorsque l'on ouvre un fichier.

7 Fonctions diverses

7.1 Fonctions de données

Il est fréquent d'avoir des fichiers contenant différents champs séparés par un délimiteur quelconque. Une fonction très utile dans ces cas là est la fonction *explode*. Sa syntaxe est la suivante

explode (« caractère délimiteur », chaîne de donnée)

Prenons un exemple avec l'extrait de fichier suivant :

Roose | Philippe | Iut Informatique | Bayonne | Informatique
Goudin | David | LaBri | Bordeaux | Calcul Parallèle

Ce fichier décrit des enseignants avec leur nom, prénom, lieu de poste, matière principale. Si l'on souhaite afficher ce fichier d'une manière « plus jolie », il est nécessaire de récupérer champs après champs et d'afficher chacun d'eux comme désiré. Sans la méthode *explode*, il serait nécessaire de récupérer caractère par caractère, tester si le séparateur est atteint, si oui, …
explode réalise ce travail pour nous en retournant dans un tableau l'ensemble des champs. Il ne reste plus qu'à « boucler » sur ce tableau pour récupérer les valeurs et les traiter.

Il est à noter qu'il existe sa réciproque, la fonction *implode* qui retourne une chaîne et qui accepte en paramètre un tableau et un délimiteur (qui peut être lui même une chaîne).

L'ensemble des fonctions « classiques » rencontrées en C sur les chaînes se retrouvent également en PHP. On y retrouve ainsi : *strlen, strcmp, strcasecmp*(idem *strcmp*, mais respecte la casse), mais aussi :

- *strpos (chaîne, sous chaîne)* : retourne la position de la sous chaîne dans la chaîne. Dans le cas où la chaîne existe en plusieurs exemplaires, c'est la position de la première occurrence qui est retournée. *strrpos* retourne quand à elle la position de la dernière occurrence.
- *strstr (chaîne, sous chaîne)* retourne la portion de la chaîne à partir de la première occurrence de la sous chaîne.
- *foreach (nom tableau)* : A chaque appel, cette fonction retourne la valeur suivante du tableau.
- *strlen (chaîne)* : retourne la taille de la chaîne.
- *strtolower|strtoupper (chaîne)* : retourne la chaîne passée en paramètres an minuscules (resp. majuscules).
- *str_replace (car d'origine, car de destination, chaîne)* : remplace le caractère d'origine par le caractère de destination dans la chaîne.
- *trim(chaîne)* : supprime les caractères invisibles (espaces,\n, …) au début et à la fin de la chaîne.
- *ereg(chaîne à chercher, chaîne)* : retourne vrai si la chaîne à chercher (sous forme de chaîne ou sous forme d'expression régulière) est contenue dans chaîne.

7.2 Fonctions mathématiques

On retrouve en PHP l'ensemble des fonction mathématiques que l'on retrouve en C, à savoir :
Abs, cos, sin, tan, acos, …, exp, sqrt, … Ces fonctions acceptent un nombre en paramètre et retournent, la valeur absolue, le cosinus, le sinus, la tangente, l'arc cosinus, …, le carré, la racine carré, …
Il existe également une fonction *pi*( ) retournant une valeur approchée de PI.

7.3 Gestion de date, heure, temps

Dans ce domaine également, PHP fourni un ensemble de fonctions particulièrement appréciables comme *date(format)*.

Les codes des formats de date sont les suivants :

Code	*Description*
a	am ou pm
A	AM ou PM
d	Jour du mois avec suppression des 0
D	Jour de la semaine, abrégé en trois lettres
F	Nom du mois
h	Heure, de 1 à 12
H	Heure, de 0 à 24
i	Minutes
j	Jour du mois, avec conservation des 0
l	Jour de la semaine
m	Chiffre du mois
M	Abréviation du nom du mois
S	Suffixe ordinal pour le jour du mois
U	Nombre de secondes depuis le 1/1/1970
y	Année, sur 2 unités
Y	Année, sur 4 unités
z	Jour de l'année

Exemple d'utilisation :

```
< ?
$ladate=date(‘’ l j F Y’’)
$lheure=date(‘’h : i - a’’) ;
print ‘’Nous sommes le $ladate <BR>’’ ;
print ‘’Il est : $lheure <BR>’’ ;
?>
Affichera :
Nous sommes le Thursday 16 November 2000
Il est : 08:56 - am
```

Il est une autre fonction qui peur avoir son utilisé, c'est la fonction *sleep(nombre de secondes*) qui réalise une pause dans l'exécution du script. La fonction *usleep* fait de même, mais l'unité est la miliseconde.

8 Mail

Il existe une méthode PHP permettant d'envoyer un mail directement, sans appeler un quelconque gestionnaire de courrier.
La fonction *mail* (ou *email* parfois) permet de réaliser cela. Elle nécessite au moins trois paramètres :

- Le destinataire,
- L'objet du message,
- Le corps du message.

```
< ?
mail(‘’dupont@mondomaine.fr’’, ‘’Test de la commande mail’’, ’’Voici le corps du mail’’) ;
?>
```

Enverra un mail à dupont@mondomaine.fr avec comme sujet de mail « Test de la commande mail », et comme corps du mail : « Voici le corps du mail ».

9 Manipulations d'images

Il existe toute une série de fonctions permettant de manipuler des images (au format GIF, JPEG et PNG). Elles permettent par exemple de retourner directement une image crée au cours du script PHP.

Nous n'allons pas décrire ici ces très nombreuses fonctions que nous demanderaient plus de temps que nous n'en n'avons mais nous allons décrire les principales qui nous servirons d'illustration afin de comprendre les mécanismes.

getimagesize (nom de fichier) : cette fonction est disponible avec toutes les versions de php (ce qui n'est pas les cas avec les fonctions qui suivront). Le nom de fichier doit correspondre à l'un des formats suivants : GIF, JPEG, PNG. Elle retourne un tableau de 4 éléments : [0] : largeur en pixels, [1] : hauteur en pixels, [2] : type d'image (GIF = 1, JPEG = 2, PNG = 3), [3] : chaîne au format 'HTML' du type « HEIGHT=### WIDTH=### » utilisable dans une balise IMG.

Afin que votre navigateur sache que ce sont des images qu'il va recevoir, il est nécessaire que la première information qui lui est communiquée soit le type de l'image.

```
<php
header(« Content-type : image/jpeg ») ;
…
php >
```

Une fois que le type de l'image est défini, il faut indiquer la taille de l'image en pixel par la fonction :
imagecreate(largeur, hauteur). Celle-ci retourne l'identifiant d'une image. Cet identifiant sera utilisé par la plupart des fonctions graphiques. On pensera impérativement à libérer l'espace mémoire une fois que l'image crée est terminée (*imagedestoy(identifiant image)*). La destruction d'une image n'implique pas sa disparition à l'écran.

Maintenant que nous connaissons à la fois le format et la taille de notre image, on peut choisir une couleur :
imagecolorallocate(entier image, entier rouge, entier vert, entier bleu).
Cette fonction alloue une couleur à l'image spécifiée. Chaque composant de couleur peut prendre une valeur entre 0 et 255. L'identifiant retourné servira dans d'autres fonctions pour faire référence à cette couleur.

La fonction *imagefill* (image, x, y, couleur) permet de remplir un espace par une couleur. Utilisée dès le départ, elle permet de donner un fond à une image.

```
Exemple : imagefill ($monimage, 0, 0, $macouleur) ;
```

Fonctions diverses :

- Dessiner un caractère dans une image : *imagechar($image, police de car., x, y, « car », couleur).* Le paramètre police correspond au numéro d'une des 5 polices de caractères disponibles en standard dans PHP. Les coordonnées (x,y) réfèrent au coin supérieur gauche de la lettre. La fonction *imagecharup* fait de même mais avec des caractères orientés vers le haut.
- *imagestring et imagestringup* sont identiques à la précédente mais permettent le dessin de chaînes de caractères.
- Récupérer la couleur d'un pixel : *imagecolorat(image, x, y)* retourne l'index de la couleur spécifiée aux coordonnées x,y.
- *imagecolorstotal($image)* retourne le nombre de couleurs dans l'image spécifiée.
- *imagecolortransparent(entier image, entier couleur)* rend transparente la couleur spécifiée.

```
< ?
imagefillrectangle($monimage, 30, 30, 70, 70, $macouleur) ; // dessine un rectangle dans une certaine couleur
imagecolortransparent($monimage, $macouleur) ; // rend cette couleur transparente.
?>
```

- *imagecreatefrom[jpeg|gif|png](nom du fichier)* permet de charger une image dans le format spécifier. Il est ensuite possible d'éventuellement travailler sur cette image.
- *image[jpg|gif|png] (id image)* retourne l'image vers le navigateur.
- *imagedashedline(image, x début, y début, x fin, y fin, couleur)* trace une ligne en pointillé en partant des coordonnées (x début, y début) jusqu'à (x fin, y fin). La fonction *imageline* fonctionne de la même manière et trace une ligne pleine.

- *imagefilledpolygon(image, tableau de points, nombre de points, couleur)* autorise le dessin de polygone. Les coordonnées sont contenues dans un tableau de points (2 à 2), le paramètre nombre de points indique combien de points sont à récupérer dans le tableau. La fonction *imagepolygon* réalise la même chose mais ne trace que le contour du polygone.
- *imagefilledrectangle($image, x haut gauche, y haut gauche, x bas droite, y bas droite, couleur)* trace bien évidemment un rectangle plein, alors que son pendant, *imagerectangle* ne trace que les bordures.

```
< ?
imagefilledpolygon($image, array(100,10,50,60,150,60), 3 $color) ; // trace un rectangle
?>
```

- Il est également possible de charger des polices de caractères avec *imageloadfont(nom de la police de caractère)*. Ces polices peuvent ensuite être utilisées avec la fonction *imagestring* déjà vue ci-dessus.

```
< ?
$mapolice=imageloadfont(« helvetica ») ;
imagestring($image, $mapolice, 10,10, « Coucou », $couleur ) ;
?>
```

- Plus simple, mais peut être plus utile (tracé de courbes par exemple), la fonction *imagesetpixel(image, x, y, couleur)* permet de placer un pixel dans la zone image.
- Les fonctions *imagesx(image)* et *imagesy(image)* permettent se connaître la largeur et la hauteur d'une image (souvent nécessaire pour connaître la taille d'une image chargée dynamiquement, par exemple avec *imagecreatefrompng*

10 Base de données

PHP offre un nombre de fonctions impressionnantes permettant aux scripts de récupérer des données à partir de quantité de SGBD différents. En natif, PHP gère les drivers pour accéder aux fichiers *dBase, mSQL*, *mySQL* ainsi que tous les SGBD supportant les accès via ODBC (*Open DataBase Connectivity*).
Nous n'allons pas décrire l'ensemble de ces fonctions, mais uniquement celles qui sont nécessaires pour une utilisation « classique ».

Nous allons nous baser ici sur une connexion à un SGBD distant de type *mySQL* avec une interrogation de la base en SQL.

10.1 Ouverture d'une base

Il existe essentiellement deux fonctions permettant l'accès à une base. La première, *mysql_connect* permet de se connecter au SGBD, la seconde *mysql_select* permet de sélectionner une base.

Exemple

```
<?
   $bdd= "mabase"; // Base de données
   $host= "sql.iutbayonne.univ-pau.fr";
   $user= "roose";  // Utilisateur
   $pass= "12345";

   mysql_connect($host,$user,$pass) or die ("Impossible de se connecter à la base de données");
   mysql_select_db($bdd);
?>
```

L'exemple précédent va permettre de se connecter sur le SGBD *mySQL* situé à l'adresse sql.iutbayonne.univ-pau.fr, avec une identification personnelle.
Bien évidemment, comme à l'accoutumé, lorsqu'on ouvre quelque chose en informatique, on le referme. On n'oubliera pas donc pas de fermer l'accès à la BD lorsque l'on en aura terminé avec elle, avec *msql_close( )*.

10.2 Requêtes SQL

Une fois la base de donnée ouverte, il est possible de l'interroger en langage SQL classique. Pour cela, il est nécessaire de lui transmettre la requête à l'aide de la fonction : *msql_query* dont voici un exemple d'utilisation :

```
$query =  "SELECT num, pays, date, circuit FROM $nomtable "; //$nomtable contient le nom de la table.
$result= mysql_query($query);
```

Bien qu'ici la requête est mise dans une variable, il est tout à fait possible de l'inclure directement dans la commande *mysql_query*.

Généralement après chaque requête, on teste si tout c'est bien passé. Pour cela, on appelle la fonction *mysql_error* qui récupère le dernier message d'erreur retourné par une fonction *mySQL*.

```
if (mysql_error()){  // Erreur base de données, sûrement la table qu'il faut créer
   print  "Erreur dans la base de données : ".mysql_error(); // On concatène et on affiche l'erreur produite.
   exit();
}
```

10.3 Récupération de données

La fonction *mysql_fetch_row( )* retourne un tableau qui représente tous les champs d'une rangée de résultat (un tuple). Chaque appel produit le tuple jusqu'à ce qu'il n'y en ai plus.
Il existe d'autres fonctions pour réaliser cela, mais celle-ci est la plus rapide pour obtenir des résultats à partir d'une requête.

```
while ($row=mysql_fetch_row($result)) // $result a été obtenu par le msql_query précédent.
   {
   // récupération des informations
   $num = mysql_result($rowt[0]);
   $pays = mysql_result($row[1]);
   $date = mysql_result($row[2]);
   $circuit = mysql_result($row[3]);

   print '' ... '' ;
}
```

10.4 Utilisation d'ODBC

Afin de communiquer avec divers SGBD, un logiciel client utilise une API (*Application Programming Interface*) appelée ODBC. Ces API sont généralement écrites par les éditeurs de SGBD eux mêmes. Il est ainsi possible à partir d'un même programme d'accéder à différents SGBD à partir du moment où l'on possède les drivers de son API.
ODBC utilise le langage SQL supporté par la quasi totalité des SGBD, et particulièrement, les SGBD relationnels.

Nous n'allons pas décrire l'utilisation des fonctions ODBC puisqu'elles s'utilisent de manière identiques à celles décrites précédemment pour *mySQL*.

11 Programmation objet : PHP et les classes

Avec PHP, on programme comme on veut…et même en objet si on le désire. Attention, PHP n'étant pas à la base un langage de programmation objet, il ne supporte pas tous les concepts inhérents à ce style/philosophie de programmation. Néanmoins, l'utilisation de classes bien définies permettra aux utilisateurs d'exploiter au mieux les fonctionnalités déjà développées.

11.1 Syntaxe de la programmation objet

La syntaxe est fortement inspirée du C++.

11.1.1 *Allocation d'un objet*

Tout se passe comme une allocation classique en C++, la seule différence est que l'on retrouve l'éternel $ propre au PHP et langage de scripts en général.

```
$monObjet = new maClasse(paramètres éventuels du constructeur) ;
```

11.1.2 Accès aux méthodes et attributs

```
$monObjet->nomMethode(…) ;
$monObjet->nomAttribut = … ou $mavar = $monObjet->nomAttribut
```

On fera attention à mettre le $ sur le nom de l'objet et pas sur le nom de la méthode/attribut. Lorsqu'on désirera affecter un attribut relatif à sois même (le *this*), on procédera de la même manière : *$this->*…
PHP n'ayant pas de déclaration préalable des variables nécessaires, l'oublie de *$this->* ne provoquera pas d'erreur, une nouvelle variable sera ainsi définie et utilisée, mais ce ne sera pas l'attribut de la classe.

11.1.3 Définition d'une méthode

```
function maMethode([$param1, $param2) {
  return $… // permet à la méthode de faire un retour de « ce que l'on veut »
}
```

11.1.4 Déclaration d'une classe

```
classe MaClasse {
  $attribut1 ;
 $attribut2 ;

 function maMethode1( ) {
  …
}
 function maMethode2($param1, $param2= « je suis une chaîne ») { // paramètre par défaut…
…
}
} // fin de la déclaration de la classe.
```

Communément, les fichier de déclaration comporteront l'extension *.inc* (pour *include*). Afin de ne pas surcharger le code, et dans un but de réutilisation maximum, nous inclurons le code via la commande *include(« maclasse.inc »)* dans le fichier PHP exploitant cette dernière. On procédera ainsi :

```
< ?php
  include (« maClasse.inc ») ;
?>

<HTML> <HEAD> …</HEAD> <BODY>
< ?php
$monObjet = new maClasse(…) ;
?>
</BODY></HTML>
```

11.1.5 Héritage

Et oui, et c'est souhaitable, il est également possible de profiter des avantages de l'héritage :
Classe maClasseHéritée extends maClasse {
…
}

12 Différences PHP3/PHP4

PHP/FI (Personal Home Page / Form Interpreter) fût la première version officielle de PHP (tout est parti d'une librairie Perl à l'origine). De cette version est issue directement PHP3 résultant d'une réécriture complète de PHP/FI. PHP3 et PHP/FI ne sont plus officiellement supportés, néanmoins, ils persistent encore en particulier

chez certains (la plupart ?) des hébergeurs. Enfin, PHP4 résulte à son tour d'une réécriture de PHP3 et utilise le moteur Zend. Ces modifications ont été réalisées par Zeev Zuraski, Andi Gutmans. C'est la branche courante de PHP.

Cette version propose un certain nombre de modifications :

- PHP4 : ré-écriture complète du moteur d'interprétation. D 'une interprétation ligne à ligne, on passe à une compilation du code : souhaitable pour de gros développements.
- Meilleure gestion du ' Garbage Collector ' avec l'objectif de permettre la réalisation de gros projets (c'est une conséquence directe du point précédent).
- « Bufferisation » des sorties. Dans les versions précédentes, les sorties étaient directement produite sur le périphérique correspondant. Par exemple, un *print* se traduisait immédiatement par son écho à l'écran. Il est maintenant possible d'agir sur le buffer pour ne réaliser l'affichage que lorsqu'il est nécessaire.
- Comparaison valeur à valeur & type à type : **= = =**. PHP mêlant les types, la comparaison, par exemple, du nombre 5 avec la chaîne '5' provoquait l'égalité. A l'aide de l'opérateur 3 =, la comparaison est également réalisée sur le type.

```
$intval = 5
$carval ='5'
if ($intval = $carval) // égalité
if ($intval === $carval) // non égalité
```

- Ajout d'un type Booléen
- Gestion des COM sous Windows

Exemple tiré de www.phpheaven.net

```
<?php
$word = new COM( "word.application") or die( "Impossible to instantiate WordApp");
print  "Word is running, version {$word->Version}\n<br>";
$word->Visible=1;
$word->Documents->Add();
$word->Selection->TypeText( "This is a test...");
$word->Documents[1]->SaveAs( "test_com_php.doc");
$word->Quit();
?>
```

- Facilité de production d 'HTML. Pour renvoyer du HTML en PHP3, on est souvent confronté à d'innombrables *print*. L'utilisation d'étiquettes facilite grandement ceci.

```
$maVariable = « PHP4 »
print <<< mylabel
<h1>Coucou</h1>
J 'utilise une autre technique pour faire du HTML
Gr&acirc;ce &agrave;
<p>$maVariable </p> et l 'utilisation d '&eacute;tiquettes.<br>
mylabel;
```

- Gestion des sessions : évite de passer par des variables partagées et/ou des cookies. Les variables de session sont stockées dans un SGBD-R et/ou dans un fichier ASCII, mais contrairement aux cookies, elles sont stockées sur le serveur. Seul l'indentifiant de la connexion est stocké sur le poste du client. Les fonctions associées sont les suivantes :
  - *bool session_start( ).* Dans le cas ou true est retournée, cette fonction permet soit de créer une session (si pas encore définie) soit d'utiliser les variables de la session en cours.
  - *string session_id ([string id])* permet d'accéder à l'identifiant unique d'une session. Dans le cas où un paramètre est précisé, l'identifiant de session changera et prendra la valeur fournie
  - *string session_name([string name])* retourne le nom de la session en cours ou le change avec la valeur du paramètre précisé.

- bool session_register (mixed name [, mixed ...]) permet de créer des variables de session (il faut que session_start() soit en entête de page).

```
if (!isset($Var1)) {
session_register("Var1");
ou
if (!session_is_registered("Var1")) {
session_register("Var1","Var2","Var3", … ); // permet de déclarer plusieurs variables en même temps.
```

- *bool session_unregister (string name)* permet de supprimer une variable de session et donc de libérer par la même occasion de l'espace mémoire. *session_unset( )* supprime toutes les variables de la session. Enfin, *session_destroy( )* détruit la session courante.

13 PHP vs ASP

Dans l'absolu, PHP et ASP ont une approche identique. Le code est intégré dans les pages Web et est interprété sur le serveur (contrairement à JAVA et à JAVAScript par exemple.
Lorsque seul PHP3 existait, l'avantage en terme de rapidité était aux ASP. De plus ils gèrent depuis le départ les variables de session alors qu'il aura fallu attendre PHP4 pour les avoir. Néanmoins, des solutions en PHP3 existaient (et existent toujours, la PHPLib : http://phplib.netuse.de/). L'avantage allait également sur le nombre de fonctions présentes en PHP3…l'avantage s'est accru avec PHP4.
L'autre avantage à PHP est que les ASP sont très largement liées au monde Microsoft. Des solutions pour Unix ont été développées (ChiliSoft : http://www.chilisoft.com/) mais ne sont pas aussi complètes. PHP est disponible sur à peu près toutes les plate-formes…et gratuitement.
Enfin, PHP peut fonctionner avec la plupart des SGBD (Oracle, MySQL, Sybase, …) alors que les ASP sont liés au SGBD supportant ODBC ce qui les rend qui plus est moins performants.

14 Conclusion

Nous n'avons décrit ici que les fonctions de base permettant de débuter en PHP. Mais, bien qu'encore très jeune, ce langage possède une multitude de fonctions, impossibles à décrire ici, facilitant la vie du développeur. On pourra se référer par exemple à un bon livre de référence qui a servi à l'élaboration de ce support de cours.

Maîtrisez PHP4 – Wankyu et al. - Éditions Campus Press/Wrox – 2001 – ISBN 2-7440-9002-6
Programmez en PHP – Léon Atkinson – Éditions Campus Press Référence – 1999 – ISBN 2-7440-0771-4